

Support Vector Machine Tutorial

Contents

1. Introduction to Support Vector Machine (SVM)	2
2. Data Preparation and Exploration	2
2.1 Data Preprocessing and Exploratory Data Analysis	3
3. Data Splitting	6
4. Model Training and Evaluation	7
5. Decision Boundary Visualization	9
6. Conclusion	10
References	11

Figure 1: Code for loading dataset	2
Figure 2: First five row of the dataset	3
Figure 3: Dataset Information	3
Figure 4: Check the missing values	4
Figure 5: Statistical Summary of the dataset	4
Figure 6: Scatter plot of the dataset	5
Figure 7: Class distribution of the dataset	6
Figure 8: Feature extraction and data splitting	7
Figure 9: Confusion matrix on training data and test data	8
Figure 10: Accuracy score on training and test data	8
Figure 11: Classification report on training data and test data	9
Figure 12: Decision Boundary Visualization	10

1. Introduction to Support Vector Machine (SVM)

Support Vector Machines (SVMs) are powerful supervised learning algorithms used for classification and regression tasks. The core principle of SVM is to find an optimal hyperplane that separates data points of different classes with the maximum possible margin. This margin maximization makes SVM robust and effective, especially in high-dimensional spaces [1]. SVMs can be particularly useful in binary classification problems, where they seek to identify a decision boundary that best separates two classes [2].

One of SVM's strengths is its flexibility in handling linear and non-linear data through the use of kernel functions. Kernels transform the original input space into a higher-dimensional feature space, enabling SVM to classify non-linearly separable data. Common kernel functions include **Linear Kernel** that is suitable for linearly separable data [3], **Radial Basis Function (RBF) Kernel** that is a popular choice for non-linear problems due to its flexibility. **Polynomial Kernel** that is useful when the relationship between features is polynomial in nature. **Sigmoid Kernel** that is often used in neural networks.

This tutorial demonstrates the implementation of SVM on a dataset, highlighting different kernel types and evaluating their impact on model performance.

2. Data Preparation and Exploration

The dataset used in this tutorial is a 2D dataset [4] for the implementation and better explanation of support vector machine. 'Aggregate.txt' dataset file has been used in this tutorial. The dataset has 2 dimensions of data points and corresponding label. The dataset is loaded using the pandas library in the notebook. Each row in dataset represents a data point in a 2D space, with the label indicating its class. Figure 1 shows the code snippet of loading the dataset into the notebook.

Load the Dataset

```
[ ] df = pd.read_csv('/content/Aggregation.txt', sep = '\t', skiprows=7, header = None, names=['x1', 'x2', 'y'] )
```

Figure 1: Code for loading dataset

2.1 Data Preprocessing and Exploratory Data Analysis

▼ **Fisrt Five Rows of Dataset**

```
df.head()
```

	x1	x2	y
0	15.55	28.65	2
1	14.90	27.55	2
2	14.45	28.35	2
3	14.15	28.80	2
4	13.75	28.05	2

Figure 2: First five row of the dataset

After loading the dataset, first we should see the structure of the dataset. As shown in figure 2, the code snippet is used to display the structure of the dataset by printing first five rows of the dataset. The structure in figure 2 indicates that the dataset has two features **X1**, **X2**, and one target variable 'y'.

▼ **Information About Dataset**

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 788 entries, 0 to 787  
Data columns (total 3 columns):  
#   Column  Non-Null Count  Dtype  
---  ---      -  
0   x1       788 non-null    float64  
1   x2       788 non-null    float64  
2   y        788 non-null    int64  
dtypes: float64(2), int64(1)  
memory usage: 18.6 KB
```

Figure 3: Dataset Information

After exploring the dataset, we should explore other information of the dataset. Figure 3 shows the code snippet to display the information of the dataset. Figure 3 indicates that the dataset has 3 columns and 788 entries (rows or records). It also indicates the name of the columns, non-null count, and datatype of the features and target variable. In our case, features **X1** and **X2** have 788 non-null count, and float datatype. Similarly, the target variable **y** has 788 non-null counts, and integer datatype.

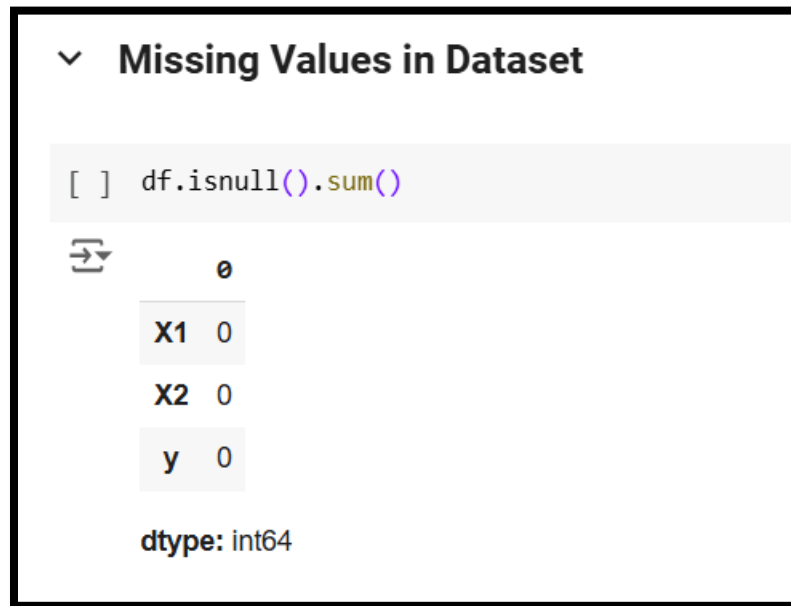


Figure 4: Check the missing values

Once we get the information of the dataset, the next step is to check for the missing values in the dataset. Figure 4 shows the code snippet to check the missing values in the dataset. Figure 4 indicates that the dataset has no missing values.

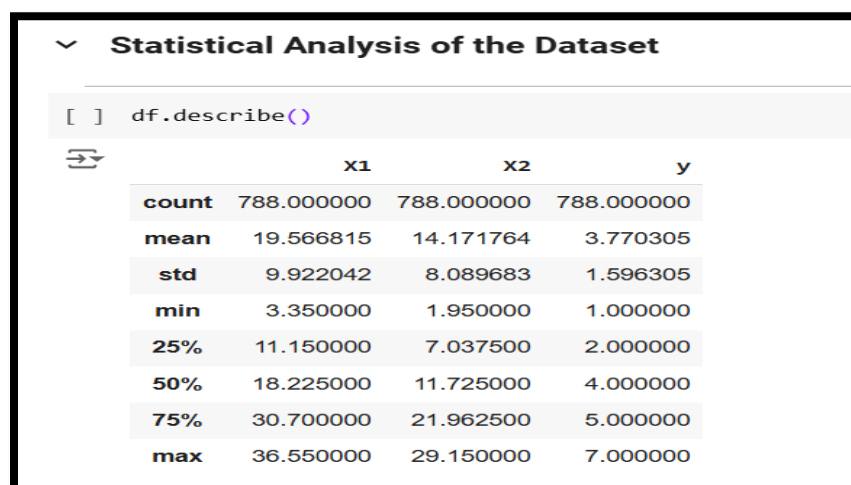


Figure 5: Statistical Summary of the dataset

Once missing values are checked, after that we should explore the statistical summary of the dataset. Figure 5 shows the code snippet to display the statistical summary of the dataset. Statistical summary includes mean, standard deviation, minimum value, 25% mean, 50% mean, and 75% mean, and the maximum value of each column of the dataset, as can be seen in figure 5.

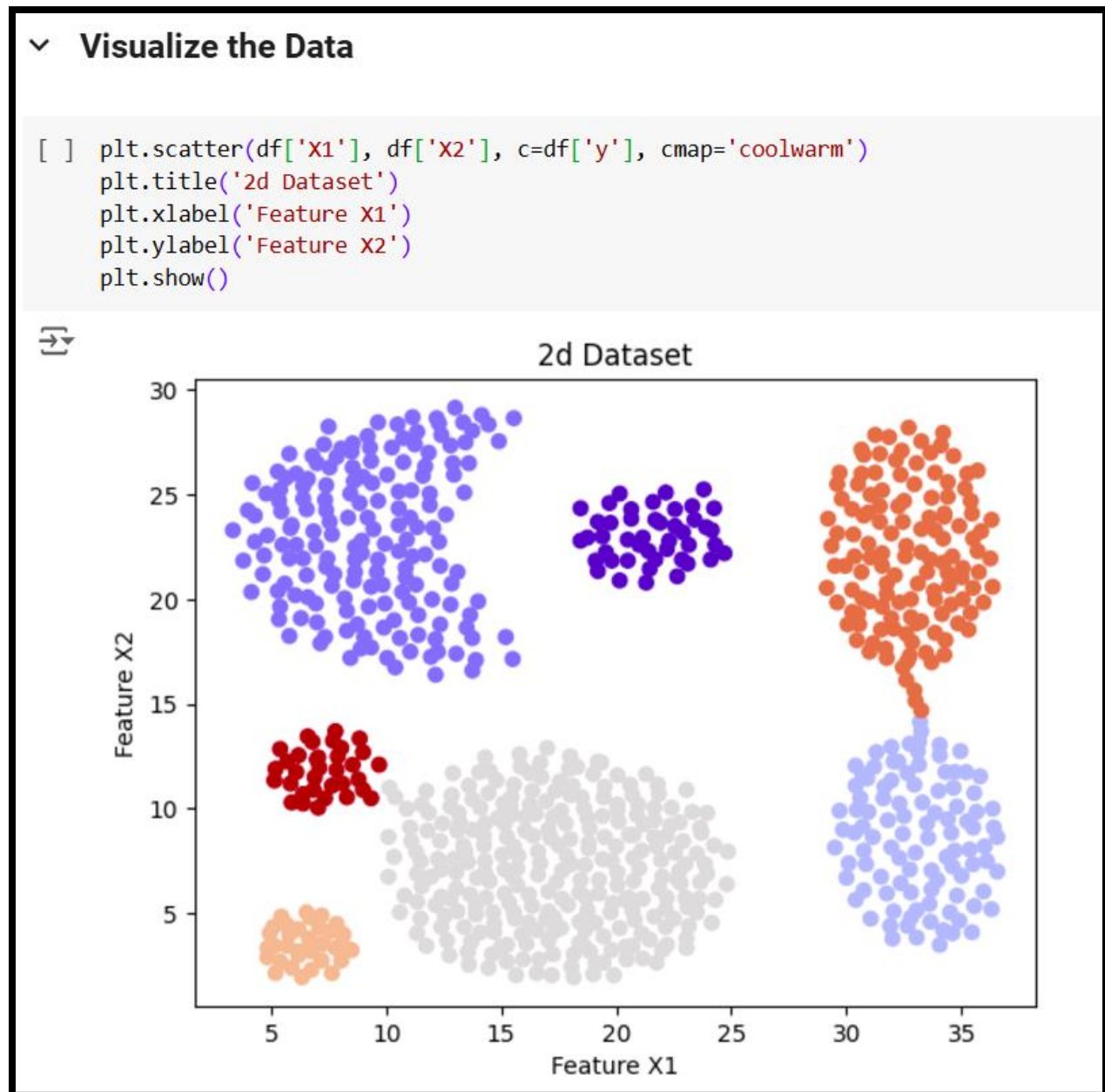


Figure 6: Scatter plot of the dataset

After exploring the statistical summary of the dataset, we should investigate the distribution of the dataset by plotting the features and target variable. The most common way to do this is to scatter plot the features along with the target variable. Figure 6 shows the scatter plot of the dataset. The scatter plot has feature **X1** at the x-axis and the feature **X2** at the y-axis. Different colors in figure

6 is due to the target variable and it shows that the data in our dataset is distributed into six classes or clusters.

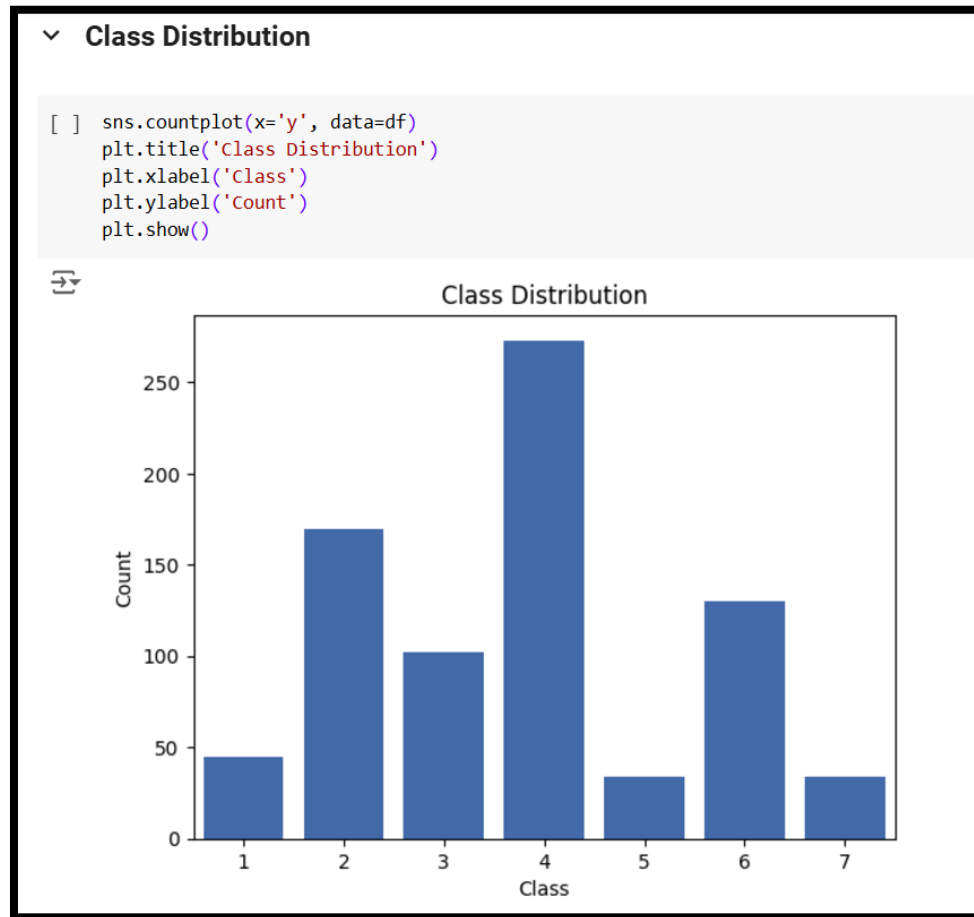


Figure 7: Class distribution of the dataset


Once the dataset distribution is explored through scatter plot, we should investigate the class distribution of the dataset. Figure 7 shows the code snippet used to plot the class distribution of the dataset. The plot in figure 7 is count plot or bar plot, it has count on y-axis and class at x-axis. It shows that there are six classes in the dataset or six labels in the target variable. Class 4 has the most samples of the dataset and class 5 has the least samples of the dataset.

3. Data Splitting

Before splitting the dataset, we should extract the features and target variable from the dataset. Figure 8 shows the code snippet used for extracting the features and target variables from the dataset. This extraction is a part of data preprocessing. Similarly, data splitting is a part of the data preprocessing.

After extracting features and target variable from the dataset, we should split the features and target variable into training and test data. This is also known as data preparation for model training.

and testing. To evaluate the model's performance, the data is split into training and testing sets using a utility of scikit-learn python library known as train-test-split. The training set (70% of the data) is used to fit the model, while the testing set (30%) evaluates its generalization ability.



```

▼ Extract features from dataset

[ ] X = df.drop('y', axis=1)
    X.shape

(788, 2)

▼ Extracting target labels from dataset

▶ Y = df['y']
  Y.shape

(788,)

▼ Split features and labels into training and test sets

[ ] x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

[ ] print(f"x_train shape: {x_train.shape}")
    print(f"x_test shape: {x_test.shape}")
    print(f"y_train shape: {y_train.shape}")
    print(f"y_test shape: {y_test.shape}")

x_train shape: (551, 2)
x_test shape: (237, 2)
y_train shape: (551,)
y_test shape: (237,)
```

Figure 8: Feature extraction and data splitting

The purpose of the data splitting is to make sure that model does not overfit and generalizes itself over unseen data. In figure 8, shape of the training and testing data have been displayed to make sure that data split successfully.

4. Model Training and Evaluation

A linear kernel is suitable for data that can be separated by a straight line. The SVM model is trained using the SVC class from scikit-learn. Once model is trained, we should get the predictions on both training data and test data. Prediction on training data is used to evaluate how well data is fit into the model, while prediction on test data is used to evaluate how well model generalizes itself on unseen data.

There are several metrics to evaluate the training and test performance of the model. We can access these metrics from scikit-learn library. Figure 9 represents the confusion matrix on training data and test data. This confusion matrix is plotted using predictions of model on the training data and test data. In figure 9, x-axis has predicted label and y-axis has true labels. There diagonal values of the confusion matrix represent the correct prediction of the model while



Figure 9: Confusion matrix on training data and test data

values in the upper triangle and lower triangle show the wrong predictions of the model. In figure 9, the data is successfully fit to the model and model has predicted the correct labels on data set. Similarly, the model has generalized well on unseen data as it has predicted one wrong label in the test data.

✓ **ii) Accuracy score for training dataset and test dataset**

```

train_accuracy = accuracy_score(y_train, ypred_train)
test_accuracy = accuracy_score(y_test, ypred_test)
print(f"Train Accuracy: {train_accuracy*100:.2f}%")
print(f"Test Accuracy: {test_accuracy*100:.2f}%")

```

➡ Train Accuracy: 100.00%
Test Accuracy: 99.16%

Figure 10: Accuracy score on training and test data

Figure 10 represents the code snippet used to compute accuracy score of the model on the training data and test data. Figure 10 shows that model has 100% accuracy score on training data which have seen in confusion matrix as well. Similarly, the model has 99.16% test accuracy which we

have seen in confusion matrix. It means that data is not fit well into model but model has generalized itself for unseen data.

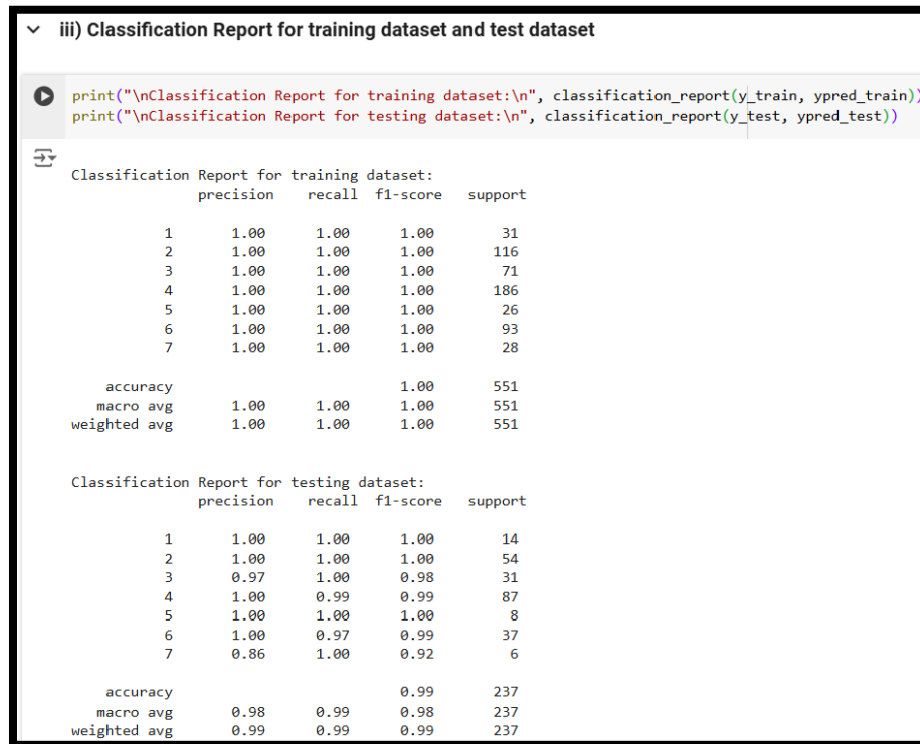


Figure 11: Classification report on training data and test data

Classification report is another metric to measure the performance of the model. It is particularly useful when datasets have uneven class distribution. The classification report has precision, recall, F1 score and accuracy. Precision is defined as the ration of true positives and predicted positives. Predicted positives are the sum of true positives and false positives that model have predicted. Similarly, recall is defined as the ratio of the true positive and actual positives. Actual positives are the sum of true positives and negative false. F1 score is defined as the harmonic mean of the precision and recall. A higher F1 score shows the model is performing better on training and test data. Figure 11 indicates that the model has 100% F1 score for training data. While it has 90-100% F1 score on test data. In summary, classification report shows that model is trained and generalized itself well.

5. Decision Boundary Visualization

Figure 12 shows a visualization of the SVM decision boundary in a 2D feature space. Mesh Grid creates a grid of points across the feature space using mesh grid utility of the numpy library, extending slightly beyond the range of the dataset features. This grid acts as a canvas for plotting the decision boundaries.

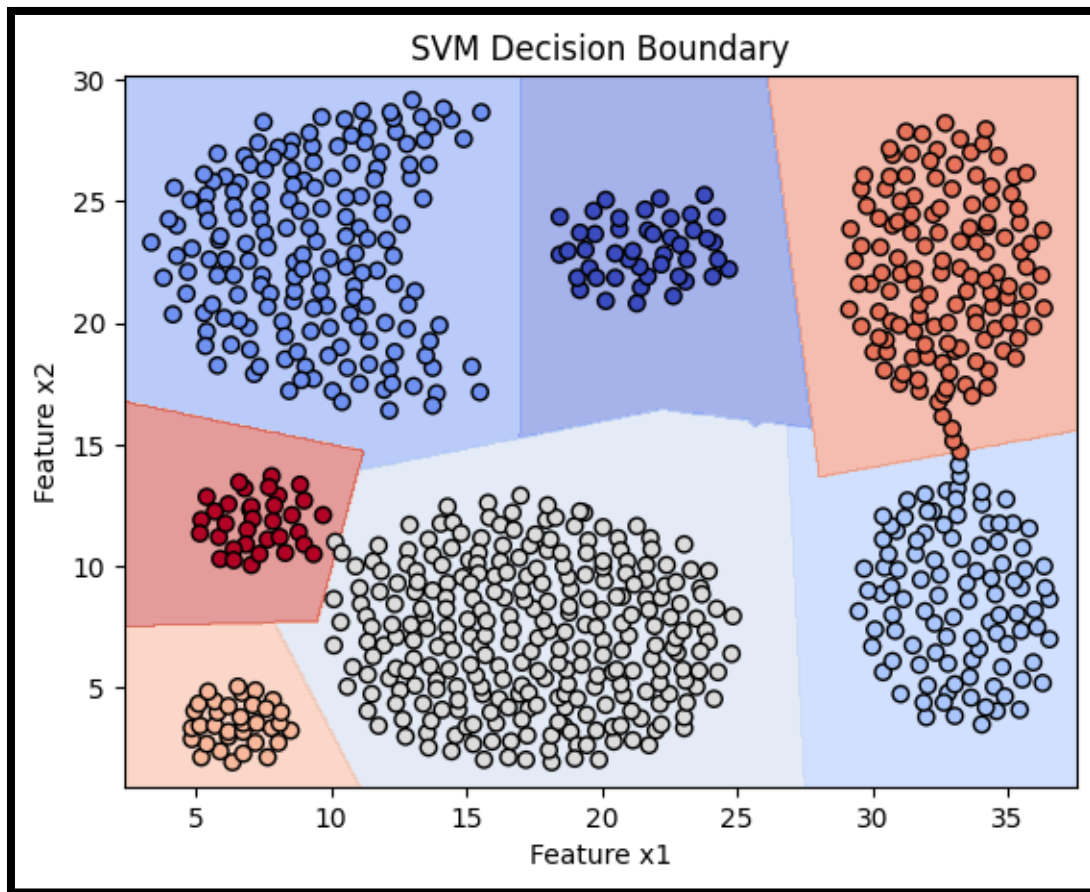


Figure 12: Decision Boundary Visualization

The trained SVM model predicts class labels for each point in the grid, creating a map of decision regions. A contour plot is used to display the decision boundaries by filling the regions between different classes with distinct colors. The original data points are plotted on top of this using a scatter plot, allowing a clear view of how the SVM separates different classes.

This visualization helps in understanding how the SVM classifies data and how the decision boundary adapts based on the chosen kernel.

6. Conclusion

Support Vector Machines are versatile and effective for classification tasks. The choice of kernel plays a critical role in the model's performance. This tutorial demonstrated the data preprocessing, exploratory data analysis, splitting the data, training and evaluation of model in detail, and how SVM model classified the data using linear kernel. For practitioners, understanding kernel functions and parameter tuning is essential for building robust models.

References

- [1] C. & V. V. Cortes, "Support-vector networks.," *Machine Learning*, vol. 20, no. 3, pp. 273-297, 1995.
- [2] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-167, 1998.
- [3] B. & S. A. J. Schölkopf, "Learning with kernels: Support vector machines, regularization, optimization, and beyond," *MIT Press.*, 2002.
- [4] H. C. a. D. Yeung, "Robust path-based spectral clustering," *Pattern Recognition*, vol. 41, no. 1, pp. 191-203, 2008.